Executing distributed applications on virtualized infrastructures specified with the VXDL language and managed by the HIPerNET framework

Guilherme Koslovski, Tram Truong Huu, Johan Montagnat, Pascale Vicat-Blanc Primet

Executing distributed applications on virtualized infrastructures specified with the VXDL language and managed by the HIPerNET framework

Guilherme Koslovski[1], Tram Truong Huu[2],
Johan Montagnat[2], and Pascale Vicat-Blanc Primet[1]

[1] INRIA - University of Lyon, France
guilherme.koslovski@ens-lyon.fr, pascale.primet@inria.fr
[2] CNRS - I3S, Sophia Antipolis, France
tram@polytech.unice.fr, johan@i3s.unice.fr

Abstract. With the convergence of computing and communication, and the expansion of cloud computing, new models and tools are needed to allow users to define, create, and exploit on-demand virtual infrastructures within wide area distributed environments. Optimally designing customized virtual execution-infrastructure and executing them on a physical substrate remains a complex problem. This paper presents the VXDL language, a language for specifying and describing virtual infrastructures and the HIPerNET framework to manage them. Based on the example of a specific biomedical application and workflow engine, this paper illustrates how VXDL enables to specify different customized virtual infrastructures and the HIPerNET framework to execute them on a distributed substrate. The paper presents experiments of the deployment and execution of this application on different virtual infrastructures managed by our HIPerNet system. All the experiments are performed on the Grid'5000 testbed substrate.

Key words: Virtual Infrastructure as a service, resource virtualization, application mapping, graph embedding problem, workflow language, topology language

1 Introduction

The convergence of communication and computation portrays a new vision of the Internet. It is becoming a worldwide cloud increasingly embedding the computational and storage resources that are able to meet the requirements of emerging applications. This resulting vision of a global facility, that brings together distributed resources to build large-scale computing environments, recalls and extends the promising vision of Grid computing, enabling both data-intensive and computing-intensive applications. In this context, the concept of virtualization is a powerful abstraction. It enables an efficient separation between the service and application layers on one hand and the physical resources layer on

the other hand. The OS-level virtual machines paradigm is becoming a key feature of servers, distributed systems, and grids. It simplifies the management of resources and offers a greater flexibility in resource usage. Each Virtual Machine (VM) a) provides a confined environment where non-trusted applications can be run, b) allows establishing limits in hardware-resource access and usage, through isolation techniques, c) allows adapting the runtime environment to the application instead of porting the application to the runtime environment (this enhances application portability), d) allows using dedicated or optimized OS mechanisms (scheduler, virtual-memory management, network protocol) for each application, e) enables applications and processes running within a VM to be managed as a whole. Extending these properties to network resources (links and equipments) through the concept of "virtual infrastructure", the abstraction of the hardware enables the creation of multiple, isolated, and protected organized aggregates on the same set of physical resources by sharing them in time and space. The virtual infrastructures are logically isolated by virtualization. The isolation also provides a high security level for each infrastructure. Moreover, virtualizing routers and switching equipments enables the customization of packet routing, packet scheduling, and traffic engineering for each virtual network crossing it.

However, programming applications on large-scale distributed environments is difficult. Defining the optimal infrastructure to execute them is another issue. The flexibility offered by virtual infrastructures could make the problem even more complex. Promising work on workflow has been done in the area of application development to optimize their usage of distributed environments. This paper proposes to explore how this work can also benefit to the composition of virtual infrastructures.

The rest of the paper is structured as follows. In section 2, we define our model of customized *Virtual Private eXecution Infrastructures* named VPXI. To specify these VPXIs we define a description language for VPXI specification and modeling, *Virtual eXecution Description Language*, VXDL. Section 3 details the process for mapping an application on physical resources in a virtualized-infrastructure context. In section 4 we illustrate the application mapping through an example with the Bronze Standard workflow. In section 5, we develop our combined network and system virtualization approach embedded in the HIPerNet software and report the experiments on a real-scale testbed using the medical image analysis application. Section 6 discusses related works. Finally, conclusions and perspectives are developed in section 7.

2 The Virtual Private eXecution Infrastructure concept

2.1 The VPXI concept

We define the *Virtual Private eXecution Infrastructure* (VPXI) concept as a time-limited interconnection of virtual computing resources through a virtual private overlay network. Ideally, any user of a VPXI has the illusion that he is

using his own distributed system, while in reality he is using multiple systems, part of the global system. The resulting virtual instances are kept isolated from each others. The members of a VPXI have a consistent view of a single private TCP/IP overlay, independently from the underlying physical topology. A VPXI can span multiple networks belonging to disparate administrative domains. Users can join from any location, deploying and using the same TCP/IP applications they were using on the Internet or their intranet.

A VPXI can be formally represented as a graph in which a vertex is in charge of active data-processing functions and an edge is in charge of moving the data between vertices. A VPXI has a life time and can be requested online or reserved in advance. It is described and submitted as a request by a user. Then, if accepted by the operating framework, it exists as a descriptor and has an entry in a VPXI table until its release time. During the activation phase, the VPXI runs in the data plane and is represented in the control plane of each allocated equipment.

2.2 VXDL: VPXI Description Language

A VPXI specification comprises the recursive description of: a) individual end resources or resource aggregates (clusters) involved, b) performance attributes for each resource element (capacity), c) security attributes, d) commercial attributes, e) temporal attributes, f) elementary functions, which can be attributed to a single resource or a cluster (e.g. request of *computing* nodes, *storage* nodes, *visualization* nodes, or *routing* nodes), g) specific services to be provided by the resource (software), h) the virtual-network's topology, including the performance characteristics (typically bandwidth and latency), as well as the security, commercial, and temporal attributes of the virtual channels. Figure 1 illustrates this concept, representing a virtual infrastructure composed by the aggregation of virtual machines interconnected via virtual channels. It shows two virtual routers (vertices r_vA and r_vB) which are used to interconnect and perform the bandwidth control among the other virtual resources (vertices r_v 1 to 8). The virtual routers can independently forward the traffic of the different virtual infrastructures which share the same physical network. Each edge represents a virtual link (as l_v1 and l_v2) with different configurations, used to interconnect a pair of virtual resources.

To enable the specifications and the manipulation of these VPXI entities we propose the VXDL (Virtual Infrastructure Description Language) [9]. It allows the description not only of the end resources, but also of the virtual network's topology, including virtual routers and timeline representation. Implemented with the XML standard, VXDL helps users and applications to create or change VPXI specifications [1]. The VXDL grammar is divided in *Virtual Resources description*, *Virtual Network Topology description*, and *Virtual Timeline description*. A key aspect of this language is that these descriptions are partially

[1] More information about VXDL is provided on http://www.ens-lyon.fr/LIP/RESO/Software/vxdl

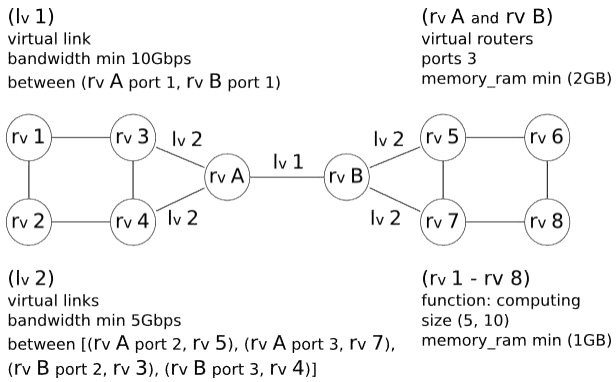

Fig. 1. Example of a VPXI composition using graph representation.

optional: it is possible to specify a simple communication infrastructure (a virtual private overlay network) or a simple aggregate of end ressources without any network topology description (a virtual cluster or grid). Below, we detail the key aspects of this language.

Virtual Resources Description: This part of VXDL grammar enables users and applications to describe, in a simple and abstract way, all the required end hosts and host groups. VXDL allows the basic resource parametrization (e.g. minimum and maximum acceptable values for RAM memory and CPU frequency). An important feature of VXDL is that it proposes cross-layer parameters. With the specification of *anchor* and *the number of virtual machines allocated per physical host* users can directly interact with lower layers and transmit application-specific information. The *anchor* parameters corresponds to a physical allocation constraint of a VPXI. Indeed, in theory a VPXI can be allocated anywhere in a virtualized substrate, but sometimes it is desirable that a virtual end host (or group) be positioned in a given physical location (e.g. a site or a machine - URL, IP) for an application-specific reason. On the other hand, in a virtualized substrate, multiple virtual machines can be allocated in the same physical host, sharing the real resources. VXDL enables the definition of a maximum number of virtual machines that must be allocated in a physical host, enabling users to interact directly with the allocation algorithm.

Virtual Network Topology Description: VXDL brings two original aspects within the network's topology description: I) the joined specification of network elements and computing elements and II) the link-organization concept, which permits a simple and abstract description of complex structures. Links can define connections between end hosts, between end hosts and groups, inside groups, between groups and VXrouters, and between VXrouters. In VXDL grammar, the definition of *source - destination pairs* for each link is proposed. The same link definition can be applied to different pairs, simplifying the specification of complex infrastructures. For example, links used to interconnect all components of an homogeneous group, as a cluster, can all be defined in a same link description. Each link can be defined by attributes such as latency, band-

width, and direction. Latency and bandwidth can be defined by the maximum and minimum values.

Virtual Timeline Description: Any VPXI can be permanent, semi-permanent, or temporary. The VPXI are allocated for a defined lifetime in time slots. Time-slot duration is specific to the substrate-management framework and consequently this parameter is configured by the manager of the environment. Often the VPXI components are not used simultaneously or all along the VPXI lifetime. Thus, the specification of an internal timeline for each VPXI can help optimizing the allocation, scheduling, and provisioning processes. Periods are delimited by temporal marks. A period can start after the end of another period or after an event.

2.3 VPXI embedding problem

Using the VXDL language, users can specify the desirable configuration and network composition of a VPXI. A VPXI request must then be interpreted, and the corresponding virtual resources have to be reserved and provisioned on available physical resources. This virtual-infrastructure allocation corresponds to a classical graph embedding problem, where the graph describing the virtual infrastructure must be mapped the physical substrate graph.

Virtual and physical graphs are of the form $G(V, E)$ where vertices V are a set of resources interconnected by a set of links (edges represented by E). Each resource or link can have a capacity represented by c_v and c_p for virtual and physical components respectively. Capacities can be interpreted as configuration of bandwidth or latency for links, and as memory size or CPU speed for resources/nodes. The information about virtual resources allocation are represented in a map notation. Each virtual component allocated in a physical one is represented as a line of map, containing the reserved capacity (c_v) and the utilization period (Δt). This time notation enables the representation of different time periods in the same VPXI, where virtual resources and links can be used in disjoined time windows, in accordance with the *timeline description* proposed by VXDL.

This embedding problem is extremely challenging and has been proved to be NP-hard. Embedding heuristics taking into account the substrate characteristics to simplify the allocation have been proposed [12, 13]. These proposals aim at maximizing the resources usage or at minimizing the maximum link load. To complement these works, we examine the virtual infrastructure description and embedding problem from the application perspective.

3 Application-mapping principles

In our model, the application-mapping process is separated in three steps:

I) workflow generation: the workflow is generated using information extracted from the application, such as benchmarks results, data input description, data

transfer in each module, and the number of nodes required to perform a satisfactory execution.

II) workflow translation into VXDL: taking into account the application's requirements (RAM configuration, CPU speed, and storage size), users can develop a VXDL description, asking for the desirable configuration of the VPXI. At this point users can also declare that some components must be allocated in a specific location as well as define the virtual network topology specifying the proximity (latency configuration) of the components and the needed bandwidth.

III) VPXI allocation: in this step VPXI management framework will allocate the virtual components respecting the configuration expressed by the user (such as parametrizations and time periods organization). In a second phase, the software configuration (OS, programming and communication tools), extracted directly from the application and described using VXDL, will be deployed within the virtual machines that compose the VPXI.

3.1 Workflow language

Complex applications able to exploit large scale distributed environments are generally described with workflows. These workflows are interpreted by engines that convert the description of work in execution scripts.

Several workflow languages have been proposed in the literature. On grid-based infrastructures, Directed Acyclic Graph (DAG)-based languages such as the MA-DAG language, part of the DIET middleware [3], have often been used. They provide a explicit, static graph of all computing tasks to be performed. To ease definition of grid applications with a complex logic to be represented, more abstract language have been introduced. For instance, Scufl was introduced within the myGrid project[2] to present data flows enacted through the Taverna workflow engine [10]. It is one of the first grid-oriented data flow languages that focuses on the application data flow rather than on the generated graph of tasks. The *GWENDIA* language[3] considered in this paper is a data-flow oriented language that aims at easing the description of the complex application data flows from a user point of view while ensuring good application performances and grid resources usage. An example of a graphic representation of workflow description is given in figure 2. In this figure *Floating* and *Reference* are representing data unit to be processed and *CrestLines*, *CrestMatch*, *PFMatchICP*, *PFRegister*, *Yasmina* and *Baladin* are processing units. *Floating* and *Reference* represent groups of data items to be processed: processing units will be invoked as many time as needed to process all data items received. The user describing the application focus on the data processing logic rather than on the execution schedule. The structural application workflow is transformed into an execution schedule dynamically, while the workflow engine is being executed.

GWENDIA is represented in XML using the tags and syntax defined below:

[2] myGrid UK e-Science project: www.mygrid.org

[3] GWENDIA is defined in the context of the ANR-06-MDCA-009 GWENDIA project: http://gwendia.polytech.unice.fr

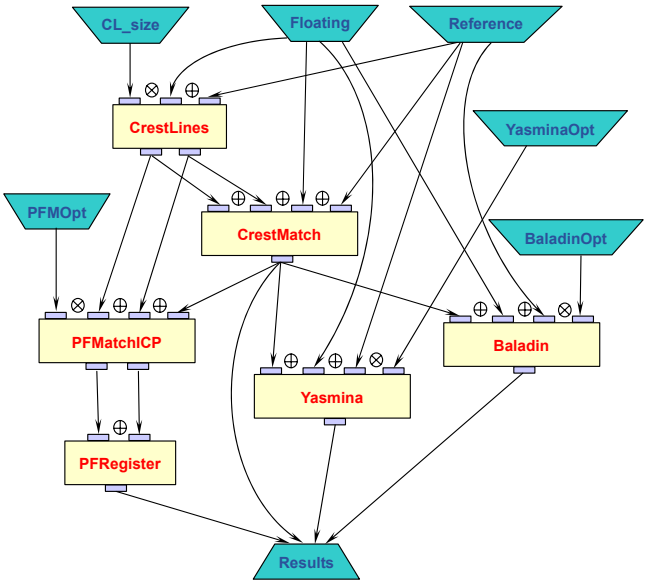

Fig. 2. Bronze Standard workflow

Types: values flowing through the workflow are typed. Basic types are `integer`, `double`, `string` and `file`.

Processors: a `processor` is a data production unit. A regular processor invokes a service through a known interface. Special processors are workflow `input` (a processor with no inbound connectivity, delivering a list of externally defined data values), `sink` (a processor with no outbound connectivity, receiving some workflow output) and `constant` (a processor delivering a single, constant value).

Processor ports: processor input and output ports are named and declared. A port may be an input (`<in>` tag), an output (`<out>` tag) or both an input/output value (`<inout>` tag). The input ports also define iteration strategies that control the number of invocation of the processor as a function of its inputs.

A simple example is given below:

```
<workflow>
  <interface>
    <constant name="parameter" type="interger">
      <value>50</value>
    </constant>
    <source name="reals" type="double"/>
    <sink name="results" type="file"/>
  </interface>
  <processors>
    <processor name="docking" type="webservice">
      <wsdl url="http://localhost/docking.wsdl" operation="dock"/>
      <in name="param" type="integer"/>
      <in name="input" type="file"/>
      <out name="result" type="double"/>
      <iterationstrategy>
        <cross>
          <port name="param"/>
          <port name="input"/>
          </cross>
      </iterationstrategy>
```

```
    </processor>
    <processor name="statisticaltest" type="diet">
      <service path="weightedaverage"/>
      <in name="weights" type="double"/>
      <in name="values" type="list(integer)"/>
      <in name="coefficient" type="double"/>
      <out name="result" type="file"/>
      <iterationstrategy>
        <cross>
          <port name="coefficient"/>
          <match tag="patient">
            <port name="values"/>
            <port name="weights"/>
          </match>
        </cross>
      </iterationstrategy>
    </processor>
  </processors>
</workflow>
```

Data link: a data link is a simple connection between a processor output port and a processor input port as exemplified below:

```
<links>
  <link from="reals" to="statisticaltest:coefficient"/>
  <link from="docking:result" to="statisticaltest:weights"/>
  <link from="statisticaltest:result" to="results"/>
</links>
```

Workflow managers are associated with these workflow language and are in charge of optimizing the execution of workflows. For example, MOTEUR [6] is a data-intensive grid-interfaced workflow manager. MOTEUR can enact a workflow represented in Scufl language or in GWENDIA language and submits the workflow tasks to a grid infrastructure. To optimize the execution, it enables three levels of parallelism: workflow parallelism, data parallelism and pipelining.

3.2 Workflow translation into VXDL

A workflow description represents the input/output data, the processors (a data-processing module), and the relationship between an application's processors. In our model, the workflow description will be translated in a VPXI description, specified in VXDL. Generally, to execute a complex application in a virtualized infrastructure, one has to consider that a middleware has to supervise the execution of the different tasks. In our example, the workflow engine (MOTEUR) and a specific task scheduler are executed for every application on independent computing resources. Input data and the intermediate results also require the presence of a file server. Therefore the VXDL description of any VPXI executing an application controled by the MOTEUR engine will contain a *generic* part describing these 3 nodes.

The *variable* part of the VPXI description directly depends on the information extracted from the workflow such as input data, the number of processors, and the links between the processors. The computation time, the data volume and the number of invocations of each module is another information that can be extracted from the workflow. Given p the number of processors (modules) of an application, the user can naively request n virtual computing resource and evenly split the set of resources among the workflow processors. Each module therefore has n/p resources. This will of course be sub-optimal since the processors have different execution times. A first variant of this naive strategy could

take into account extra information on the benchmarked execution time of each module.

4 Medical application example

Let us illustrate this VPXI description and embedding problem through a complex, real-scale medical-image analysis application known as *bronze standard*.

The bronze standard [7] technique tackles the difficult problem of validating procedures for medical-image analysis. As there is usually no reference, or *gold standard*, to validate the result of the computation in the field of medical-image processing, it is very difficult to objectively assess the results' quality. The statistical analysis of images enables the quantitative measurement of computation errors. The bronze standard technique statistically quantifies the maximal error resulting from widely used *image registration algorithms*. The larger the sample image database and the number of registration algorithms to compare with, the most accurate the method. This procedure is therefore very scalable and it requires to compose a complex application workflow including different registration-computation services with data transfer inter-dependencies.

Bronze standard's workflow is enacted with the data-intensive grid-interfaced MOTEUR workflow manager [6] designed to optimize the execution of data-parallel flows. It submits the workflow tasks to the VPXI infrastructure through the DIET middleware [3], a scalable grid scheduler based on a hierarchy of agents communicating through CORBA.

The estimated algorithm performance is valid for a typical database image. In the experiments reported below, we use a clinical database of 32 pairs of patient images to be registered by the different algorithms involved in the workflow. For each run, the processing of the complete image database results in the generation of approximately 200 computing tasks. As illustrated in figure 2, the workflow of the application has a completely deterministic pattern. All processors of this application have the same number of invocations. The execution time and the data volume transferred of each processor have been mesured in initial microbenchmarks reported in table 1.

Module	Execution time	Data volume
CrestLines	35s	32MB
CrestMatch	4s	36MB
PFMatchICP	14s	10MB
PFRegister	1s	0.5MB
Yasmina	62s	22MB
Baladin	250s	25MB

Table 1. Execution time and processed data volume for each module of bronze standard.

Let us now consider a request for a VPXI composed of 35 nodes to execute Bronze Standard's workflow. Three nodes will be dedicatd to the *generic* part : 1 node for MOTEUR, 1 node for the middleware server and 1 node for the database server. The 32 nodes left are distributed and allocated proportionally to the execution time of the workflow processors : 3 nodes for CrestLines, 1 node for CrestMatch, 1 node for PFMatchIP, 1 node for PFRegister, 22 nodes for Baladin, and 4 nodes for Yasmina. Then, for this same computing-resources set, several variants of VPXI descriptions with different network topologies can be expressed. We exemplify developing two different VPXI descriptions. The listing below presents a VXDL description of a virtual node (MOTEUR) and a computing cluster (Baladin).

```
<vxdl:resource>
 <vxdl:id>Moteur</vxdl:id>
 <vxdl:ramMemory>
   <vxdl:min>4</vxdl:min>
   <vxdl:minUnit>GB</vxdl:minUnit>
 </vxdl:ramMemory>
</vxdl:resource>
<vxdl:group>
 <vxdl:id>Cluster_Baladin</vxdl:id>
 <vxdl:function>
   <vxdl:id>computing</vxdl:id>
 </vxdl:function>
 <vxdl:size>
   <vxdl:min>22</vxdl:min>
 </vxdl:size>
 <vxdl:resource>
   <vxdl:id>Node_Cluster_Baladin</vxdl:id>
   <vxdl:ramMemory>
     <vxdl:min>2</vxdl:min>
     <vxdl:minUnit>GB</vxdl:minUnit>
   </vxdl:ramMemory>
 </vxdl:resource>
</vxdl:group>
```

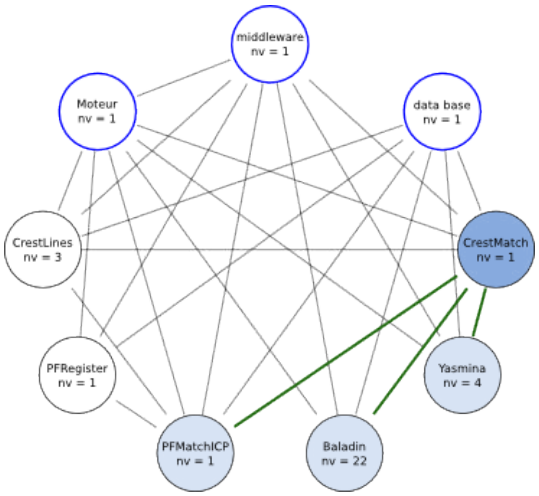

Fig. 3. VPXI description of the bronze standard's workflow.

Figure 3 illustrates the description of a VPXI using graphs. All components and network links required to execute bronze standard's workflow are represented. We developed two descriptions considering this scenario: in **VPXI 1** the

network is composed by two links type, one with low latency intra cluster and the other one with a maximum latency of 10 ms to interconnect the clusters. In **VPXI 2** the network comprises three virtual links: one with a low intra-cluster latency (maximum latency of 0.200 ms), another one with a latency of 10 ms interconnecting the components except one asking for a maximum latency of 0.200 ms to interconnect CrestMatch (dark blue) with the components PF-MatchICP, Yasmina and Baladin (blue in the figure). Listing below shows the VXDL description of this communication-intensive link.

```
<vxdl:link>
 <vxdl:id>Communication Intensive</vxdl:id>
 <vxdl:direction>bi</vxdl:direction>
 <vxdl:latency>
   <vxdl:max>0.200</vxdl:max>
   <vxdl:maxUnit>ms</vxdl:maxUnit>
 </vxdl:latency>
 <vxdl:pair>
   <vxdl:source>Cluster_CrestMatch</vxdl:source>
   <vxdl:destination>Cluster_Baladin</vxdl:destination>
 </vxdl:pair>
 <vxdl:pair>
   <vxdl:source>Cluster_CrestMatch</vxdl:source>
   <vxdl:destination>Cluster_Yasmina</vxdl:destination>
 </vxdl:pair>
 <vxdl:pair>
   <vxdl:source>Cluster_CrestMatch</vxdl:source>
   <vxdl:destination>Cluster_PFMatchICP</vxdl:destination>
 </vxdl:pair>
 <vxdl:pair>
   <vxdl:source>Database</vxdl:source>
   <vxdl:destination>Cluster_CrestMatch</vxdl:destination>
 </vxdl:pair>
</vxdl:link>
```

Let us now illustrate how each VPXI description can be embedded in a physical substrate. We propose two different solutions for both VPXI, which correspond to four different physical allocations as represented in figure 4. In this example, *Site 1* and *Site 2* represent two geographically-distributed-resources sets.

In **VPXI 1 - Allocation I**, intra-cluster link specification enables the allocation of loosely connected resources. In this embedding solution, 1 virtual machine per each physical node is allocated.

In **VPXI 1 - Allocation II** each physical node in clusters CrestMatch, PFRegister, Yasmina, and Baladin are allocated 2 virtual machines.

The **VPXI 2 - Allocation III** respects the required interconnection allocating corresponding resources in the same physical set of resources (such as a site in a grid). This embedding solution explores the allocation of 1 virtual machine per physical node.

VPXI 2 - Allocation IV explores the same physical components as *Allocation III* but allocates 2 virtual machines per physical node in the CrestMatch, PFRegister, Yasmina, and Baladin clusters.

5 Experiments in Grid'5000

To have a better insight on the influence of VPXI description, we deploy different virtual infrastructures for executing the proposed workflow in the Grid'5000 physical substrate managed and operated by the HIPerNET framework.

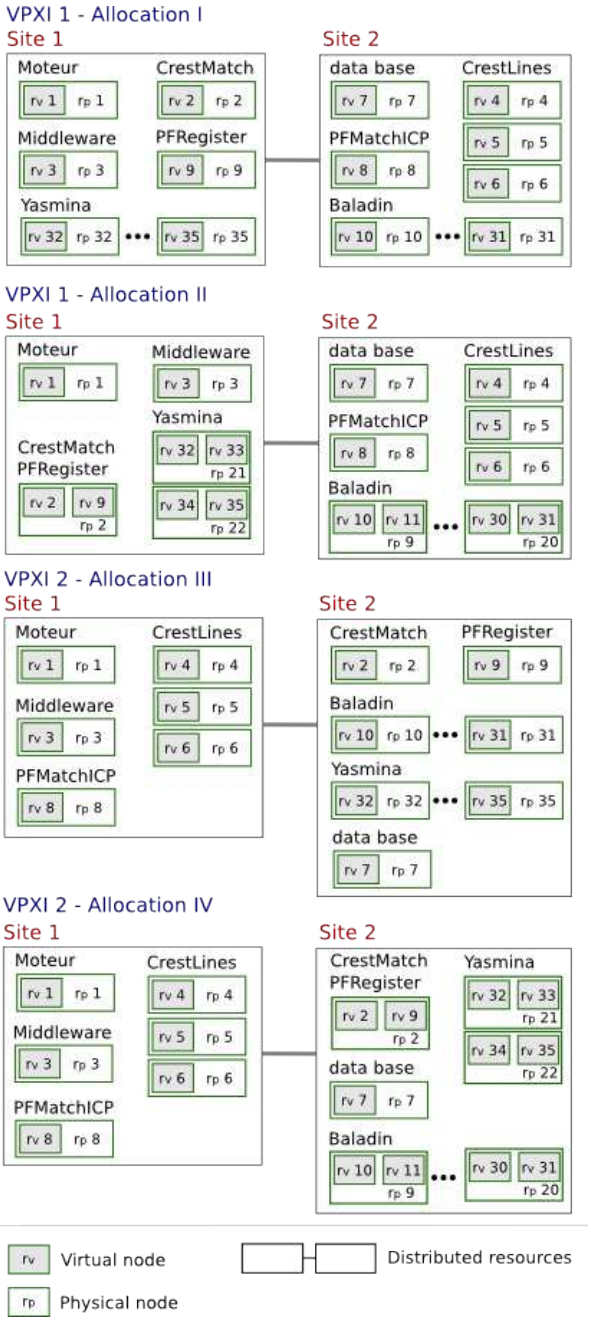

Fig. 4. Allocations of descriptions *VPXI-1* and *VPXI-2*.

5.1 HIPerNet framework and Grid'5000 substrate

The HIPerNET software[4] [11] aims to provide a framework to build and manage private, dynamic, predictable and large-scale virtual computing environments, that high-end challenging applications, like biomedical or bioinformatic applications, can use with traditional APIs: standard POSIX calls, sockets and Message Passing (MPI, OpenMP) communication libraries. With this framwork, a user preempt and interconnect virtually, for a given timeframe, a pool of virtual resources from a distributed physical substrate in order to execute his application. The originality of HIPerNet is to combine system and networking virtualization technologies with crypto-based security, bandwidth sharing and advance reservation mechanisms.

The HIPerNet substrate is transparent to all types of upper layers: upper layer protocols (e.g. TCP, UDP), APIs (e.g. sockets), middleware (e.g. Globus, Diet), applications, services and users. Hence, the HIPerNet model maintains backward compatibility with existing APIs, Middlewares and Applications which were designed for UNIX and TCP/IP APIs. Therefore, users do not need to learn new tools, developers do not need to port applications, legacy user authentication can still be used to enroll a user into a VPXI.

The HIPerNet framework aims at partitionning a distributed physical infrastructure (computers, disks, networks) into dedicated virtual private computing environment dynamically composed. When a new machine joins the physical resource set, HIPerNet prepares its operating system to enable several virtual machines (VMs) to be instantiated dynamically when required. This set of potential virtual machines is called an HIPerSpace and it is represented in the HIPerSpace Database. The HIPerSpace is the only entity that see the physical entities. A resource, volunteer to join the resource pool, is automatically initiated and registered in the HIPerSpace database. The discovery of all the devices of the physical node is also automatic. An image of the specific HIPerNet operating system is deployed on it. In our current HIPerNet implementation, the operating system image contains basically the Xen Hypervisor and its domain of administration called domain 0 (Dom 0). The HIPerSpace registrar (Operational HIPerVisor) collects and stores persistently data and manages accounts (e.g., the authentication database). It is therefore hosted by a physical machine outside of the HIPerSpace itself. For the sake of robustness and scalability, HIPerSpace registrar can be replicated or even distributed

We run the application within several virtual infrastructures created and managed by our HIPerNet software within the Grid'5000 testbed[4]. Grid'5000 enables user to request, reconfigure and access physical machines belonging to 9 sites distributed in France. In our experiment, we reserve several Grid'5000 nodes to compose a pool of physical resources that we initialize to form an HIPerSpace. To instanciate an HIPerSpace, specific tools provided by the hosted Grid are used. This is the only part aware of the physical infrastructure of the HIPerNet Software. All the other parts are independant of the physical resources

[4] http://www.ens-lyon.fr/LP/RESO/software/HIPerNET

because they use them indirectly through the services provided by HIPerNet. In Grid'5000, the HIPerSpace appears like a set of ordinary jobs scheduled by OAR with the use of a specific operating system image deployed by kadeploy

5.2 Medical imaging application deployment on the testbed

For testing VPXIs, a system image containing the operating system based on a standard Linux distribution Debian *Etch* with a kernel version 2.6.18-8 for *AMD64*, the domain-specific image processing services and the middleware components (MOTEUR and DIET) was created. The experiments on the VPXIs described in the section 4 were performed. In each experiment, we repeated the application 10 times to measure the average and standard deviation of the application makespan, the data transfer and task execution time. The physical infrastructure is reserved on the Grid'5000 clusters: *capricorne* (Lyon), *bordemer* (Bordeaux) and *azur* (Sophia) which CPUs are 2.0 GHz dual-cores Opterons. The distance between clusters is 500km and they are connected through 10Gbps links. Each VPXI is composed of 35 nodes divided in *generic* and *variable* part: 3 nodes are dedicated to *generic* part (MOTEUR, DIET, file server) using 1 CPU per node and the remaining 32 nodes of the *variable* part are allocated dependently on the VPXIs (*VPXI 1 - Allocation I* and *VPXI 2 - Allocation III* used 1 CPU per node while *VPXI 1 - Allocation II* and *VPXI 2 - Allocation IV* used 1 CPU core per node).

Coallocating resources on one grid site: the application's makespan on the *VPXI 2 - Allocation III* and *VPXI 2 - Allocation IV* is 11min 44s (±49s) and 12min 3s (±50s) respectively. This corresponds to a +3.8% makespan increase, due to the execution overhead when there are two virtual machines collocated on the same physical resource. Indeed, we present in the table 2 the average execution time of application services on the *VPXI 2 - Allocations III and IV*. We can observe that the average execution overhead is 5.17% (10.53% in the worst case and 1.28% in the best case).

Services	Allocation III	Allocation IV	variation
CrestLines	34.12 ± 0.34	36.84 ± 5.78	+7.97%
CrestMatch	3.61 ± 0.48	3.99 ± 0.63	+10.53%
PFMatchICP	11.93 ± 2.76	12.75 ± 5.35	+6.87%
PFRegister	0.78 ± 0.18	0.79 ± 0.18	+1.28%
Yasmina	59.72 ± 14.08	61.53 ± 13.98	+3.03%
Baladin	244.68 ± 16.68	247.99 ± 19.51	+1.35%

Table 2. Execution time on VPXI 2 - Allocations III and IV and 4

Resources distributed over 2 sites: when porting the application from a local infrastructure to a large scale infrastructure, the data transfer increases. Table 3 presents the data transfer time (s) of the application services on *VPXI 2 - Allocation IV* (local) and *VPXI 1 - Allocation II* (distributed over 2 sites). The

measured overhead is 150% in the worst case. Conversely, some local transfers may be slightly reduced. In the case of our application however, this overhead has little impact on the application makespan since it is compensated for by the parallel data transfer and computations. Indeed, the makespan is 12min (±12s) and 12min 11s (±20s) on *VPXI 1 - Allocation I* and *VPXI 1 - Allocation II* respectively, very similar to the performance of *VPXI 2 - Allocation IV*.

Services	Allocation IV	Allocation II	variation
CrestLines	2 ± 0.45	3.01 ± 1.6	+50.5%
CrestMatch	1.99 ± 0.34	1.83 ± 0.36	-8.04%
PFMatchICP	1.3 ± 0.4	3.25 ± 0.13	+150%
PFRegister	0.51 ± 0.23	0.43 ± 0.09	-15.69%
Yasmina	1.19 ± 0.27	1.16 ± 0.21	-2.52%
Baladin	1.17 ± 0.38	1.81 ± 1.03	+54.7%

Table 3. Data transfer time on the local VPXI 2 - Allocation IV and large scale VPXI 1 - Allocation II infrastructure

Resources distributed over 3 sites: further distributing computational resources causes an additional increase of the data-transfer overheads. An additional experiment with *VPXI 1 - Allocation II* the *generic* part of which is located in Lyon while the *variable* part is randomly distributed in Lyon, Bordeaux and Sophia leads to a makespan of 12min 13s (± 30s) with a data-transfer overhead of 176% in the worst case.

6 Related work

In this section, we briefly describe related works which explore a virtual- infrastructure composition on distributed resources, as well as the mapping process.

In [8] the authors propose the use of virtual grids to simplify application scheduling. Their descriptive language, vgDL, enables users to specify an initial description of the desirable resources, resulting in a pre-selected virtual grid corresponding to a simple vgDL description. vgDL proposes three aggregation types to specify the interconnection network: LooseBag, TightBag and Cluster. The approach proposed in VXDL is more comprehensive and allows the definition of the infrastructure's shape through the description and configuration of virtual links.

The approach of controlled virtual network infrastructures, running in parallel over a shared physical network is an emerging idea offering a variety of new features for the network. Cabo [5] proposes to exploit virtual networks for Internet Service Providers, distinguishing them from the physical infrastructure providers, and giving them end-to-end control. HIPerNET shares the same vision but focuses more on distributed computing application and proposes a language to express the infrastructure requirements in capacity, time, and space.

In [2], the authors propose VINI, a virtual network infrastructure that allows several virtual networks to share a single physical infrastructure, in a similar way to HIPerNET. VINI makes the network transparent to the user, representing each component of the network. This being one of our main interests, HIPerNET provides a language, VXDL, to specify the topology of those components. The GENI project [1] aims to build a shared infrastructure for hosting multiple types of network experiments. VXDL can help in the description of slices and HIPerNET is an orchestration framework that suits GENI's requirements.

7 Conclusion and perspectives

This paper proposed the VXDL language to specify virtual infrastructures and the HIPerNET framework to deploy and execute them. It illustrated the usage of these combined tools by a real application. In particular it developed the process of translating an applicaton's workflow into a VXDL description of a virtual private execution-infrastructure. This paper detailed the description of several virtual infrastructures for executing the same medical applications that require a high quality of service and a scalable infrastructure. Experimental results of the deployment and execution of this application in different virtual infrastructures using the HIPerNET framework within the Grid'5000 substrate assess the pertinence of the VXDL language and of the HIPerNET framework. Based on these promising results, our future works will explore an approach to automate the translation of the workflow in a VXDL description, with the aim of capitalising on the expertise of application and workflow developers to ease the embedding process while improving end-user satisfaction as well as infrastructure usage.

Acknowledgments

This work has been funded by the ANR CIS HIPCAL grant (contract ANR06-CIS-005), the French ministry of Education and Research, INRIA, and CNRS, via ACI GRID's Grid'5000 project and Aladdin ADT.

References

1. Geni design principles. *Computer*, 39(9):102–105, 2006.
2. Andy Bavier, Nick Feamster, Mark Huang, Larry Peterson, and Jennifer Rexford. In VINI Veritas: Realistic and Controlled Network Experimentation. *ACM SIGCOMM Computer Communication Review (CCR)*, 36(4):3–14, 2006.
3. Eddy Caron and Frédéric Desprez. DIET: A Scalable Toolbox to Build Network Enabled Servers on the Grid. *Int. Journal of High Performance Computing Applications*, 20(3):335–352, 2006.

4. F. Cappello, P. Primet et al. Grid'5000: A large scale and highly reconfigurable grid experimental testbed. In *GRID '05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pages 99–106. IEEE Computer Society, 2005.
5. Nick Feamster, Lixin Gao, and Jennifer Rexford. How to lease the internet in your spare time. *SIGCOMM Comput. Commun. Rev.*, 37(1):61–64, 2007.
6. Tristan Glatard, Johan Montagnat, Diane Lingrand, and Xavier Pennec. Flexible and efficient workflow deployement of data-intensive applications on grids with MOTEUR. *Int. Journal of High Performance Computing and Applications (IJHPCA)*, 22(3):347–360, August 2008.
7. Tristan Glatard, Xavier Pennec, and Johan Montagnat. Performance evaluation of grid-enabled registration algorithms using bronze-standards. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI'06)*, October 2006.
8. R. Huang, H. Casanova, and A.A. Chien. Using virtual grids to simplify application scheduling. *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 10 pp.–, April 2006.
9. Guilherme Koslovski, Pascale Vicat-Blanc Primet, and Andrea Schwertner Charão. VXDL: Virtual Resources and Interconnection Networks Description Language. In *GridNets 2008*, Oct. 2008.
10. Tom Oinn, Peter Li, Douglas B Kell, Carole Goble, Antoon Gooderis, Mark Greenwood, Duncan Hull, Robert Stevens, Daniele Turi, and Jun Zhao. *Taverna/myGrid: Aligning a Workflow System with the Life Sciences Community*, chapter 19, pages 300–319. Springer-Verlag, 2007.
11. Pascale Vicat-Blanc Primet, Jean-Patrick Gelas, Olivier Mornard, Guilherme Koslovski, Vincent Roca, Lionel Giraud, Johan Montagnat, and Tram Truong Huu. A scalable security model for enabling dynamic virtual private execution infrastructures on the internet. In *IEEE International Conference on Cluster Computing and the Grid CCGrid2009*, Shanghai, May 2009.
12. Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. Rethinking virtual network embedding: substrate support for path splitting and migration. *SIGCOMM Comput. Commun. Rev.*, 38(2):17–29, 2008.
13. Y. Zhu and M. Ammar. Algorithms for assigning substrate network resources to virtual network components. *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–12, April 2006.